Amendments to the Specification:

Please amend the paragraph (section) beginning on page 6, at line 11, as shown below:

ADPM is based on a design process modeling framework that is built on previous work and emphasizes the role of constraints. In this framework, a design is characterized by a set of variables called properties. A design property, denoted by a_i, is a variable that can take one or more values from a range $E_i = \{v^i_j, j = 1, \ldots, N^V_i\}$, where N^V_i is the total number of values a variable can achieve. Values may be numbers, strings, tuples, or complex descriptions. A property a_i to which a single value has been assigned is said to be *bound*; otherwise, it is *unbound* with an implicit value of $a_i \equiv E_i$. The properties of a correct design must satisfy a set of constraints. A design constraint is a relation, c_i, among a set of properties:

$$c_i(a_i): S_i \rightarrow \{T, F\}, \tag{1}$$

where $a_i = \{a_{i_j}, j = 1, \ldots, N^A_i\}$ denotes the arguments of c_i, and S_i denotes the cross-product of all possible argument values, *i.e.*, the design subspace restricted by c_i, and N^A_i is the total number of variables within a constraint. For example, constraint c_i, given by $P_f + P_s \leq P_M$, relates a receiver circuit's power consumption requirement, P_M, its analog front-end power, P_f, and its digital deserializer power, P_s. A constraint c_i is said to be satisfied if it holds for all combinations of the current argument values; violated if it returns false for all combinations; and consistent otherwise. The status of c_i, denoted by $s(c_i)$, indicates whether c_i is satisfied $(s(c_i) = T)$, violated $(s(c_i) = F)$, or otherwise $(s(c_i) = Unknown)$.

Please amend the paragraph (section) beginning on page 7, at line 17, as shown below:

A design process is a state-based system that goes through a series of design states. The design process history at stage n is given by $H_n=\{(<s_i, \theta_i>, i=1,\ldots,-1)\cup s_n\}$, where s_i and θ_i denote the design process state and the applied operation at stage i, respectively. Each s_i consists of: the design object hierarchy, *i.e.*, the set of all design objects currently under design, where each object is a set of properties that represent a part of the design; the design problem hierarchy, *i.e.*, the set of all formulated design problems; and the network of constraints, denoted by $C_i = \{c_i, j = 1,\ldots, N_i^C\}$, where N_i^C is the total number of design constraints. The design space at stage n is given by the cross-product of all property value ranges in s_n. A design transition, denoted by t_n, is a pair of consecutive states (s_n, s_{n+1}). s_{n+1} results from applying the next-state function, δ, to s_n:

$$[[\ s_{n+1} = \delta(s_n, \phi_n),\ ]] \tag{2}$$

$$\underline{s_{n+1} = \delta(s_n, \theta_n),}$$

where θ_n is the operation executed at stage n. The function δ applies θ_n's operator to a problem in s_n, and updates the state to s_{n+1}. δ's implementation depends on how the design process is managed.

Please amend the paragraph (section) beginning on page 10, at line 25, as shown below:

Another helpful heuristic based on existing constraint satisfaction heuristics is to execute operations that target properties connected to many constraints. It is intended to help focus first on very "constrained" properties. In ADPM, designers can apply this heuristic as they receive information about: a) constraints involved in each design problem; and b) constraints where each property appears. To help apply this heuristic, one associates a variable, denoted by β_i, with each property a_i. β_i is the number of constraints where a_i

appears: [[$\beta_i = \left|\{c_j \middle| a_i \varepsilon a_{j}\}\right|$.]] $\beta_i = \left|\{c_j \middle| a_i \in \boldsymbol{a}_j\}\right|$. Extensions of this heuristic are possible. Specifically, β_i may also include constraints indirectly related to a_i by an intermediate constraint.

Please amend the paragraph (section) beginning on page 11, at line 7, as shown below:

Timely constraint violation information allows backtracking to start early. It can also be used as the basis of a heuristic for fixing violations; specifically, to modify values of properties connected to many violations. This heuristic may help resolve multiple conflicts with a single operation and thus exit the infeasible part of the design space fast. ADPM supports this heuristic by providing designers with the following information: a) for each problem, all conflicts affecting any of its properties; and b) for each property, all conflicts where the property is involved. To help apply this heuristic, one associates a variable, denoted by α_i, with each property a_i. α_i is the number of violated constraints where a_i appears:

$$[[\ \alpha_i = \left|\{c_j \middle| (a_i \varepsilon\, a_j)\wedge(s(c_j) = F)\}\right|\ ]] \qquad (3)$$

$$\alpha_i = \left|\{c_j \middle| (a_i \in \boldsymbol{a}_j) \wedge (s(c_j) = F)\}\right|$$